

Chapter 18

Agent Intrinsic Safety: Threats on AI Brain

The intrinsic safety of an AI agent concerns vulnerabilities within the agent’s internal architecture and functionality. AI agents, by their nature, consist of multiple components: a central “brain” (the LLM), and auxiliary modules for perception and action [66]. While this modularity enables sophisticated reasoning and autonomous decision-making, it also expands the potential attack surface, exposing the agent to various internal vulnerabilities that adversaries can exploit [1130].

Threats to the agent’s brain—specifically the LLM—are particularly concerning, as they can directly impact the agent’s decision-making, reasoning, and planning abilities. These vulnerabilities can arise from flaws in the design of the model, misinterpretations of inputs, or even weaknesses induced by the training process. Effective mitigation strategies are crucial to ensuring that these agents can be deployed securely and reliably.

18.1 Safety Vulnerabilities of LLMs

The LLM, as the core decision-making component of the agent, is highly susceptible to a range of safety threats. Its central role in reasoning and action selection makes it an attractive target for adversaries. In the context of AI agents, the vulnerabilities inherent in the LLM itself are often amplified, as these models are required to function within dynamic, real-world environments where adversaries can exploit weaknesses [1131, 1132].

18.1.1 Jailbreak Attacks

Jailbreaks circumvent the safety guardrails embedded in AI agents, compelling their decision-making process to be harmful, unethical, or biased [1233]. These attacks exploit the inherent tension between an LLM’s helpfulness and its safety constraints [1134].

Formalization. To formally characterize the risks posed by jailbreaks, we analyze the probability distribution governing an autoregressive LLM’s output. For an autoregressive LLM, the probability of generating an output sequence $\mathbf{y} = \mathbf{x}_{n+1:n+m}$, given an input sequence $\mathbf{x}_{1:n}$ is modeled as:

$$p(\mathbf{y}|\mathbf{x}_{1:n}) = \prod_{i=1}^m p(\mathbf{x}_{n+i}|\mathbf{x}_{1:n+i-1}) \quad (18.1)$$

where m denotes the total length of the generated sequence. Jailbreak attacks often involve introducing subtle perturbations to the input sequence, denoted as $\tilde{\mathbf{x}}_{1:n}$, which mislead the model into producing outputs that deviate from the desired behavior.

The impact of a jailbreak attack is evaluated through its effect on the alignment reward $\mathcal{R}^*(\mathbf{y}|\mathbf{x}_{1:n}, \mathcal{A})$, which measures how closely the model’s output aligns with a set of human-defined safety or ethical guidelines, denoted as \mathcal{A} . The adversary’s goal is to minimize this reward, formalized as:

$$\mathbf{y}^* = \arg \min_{\mathbf{y}} \mathcal{R}^*(\mathbf{y}|\tilde{\mathbf{x}}_{1:n}, \mathcal{A}) \quad (18.2)$$

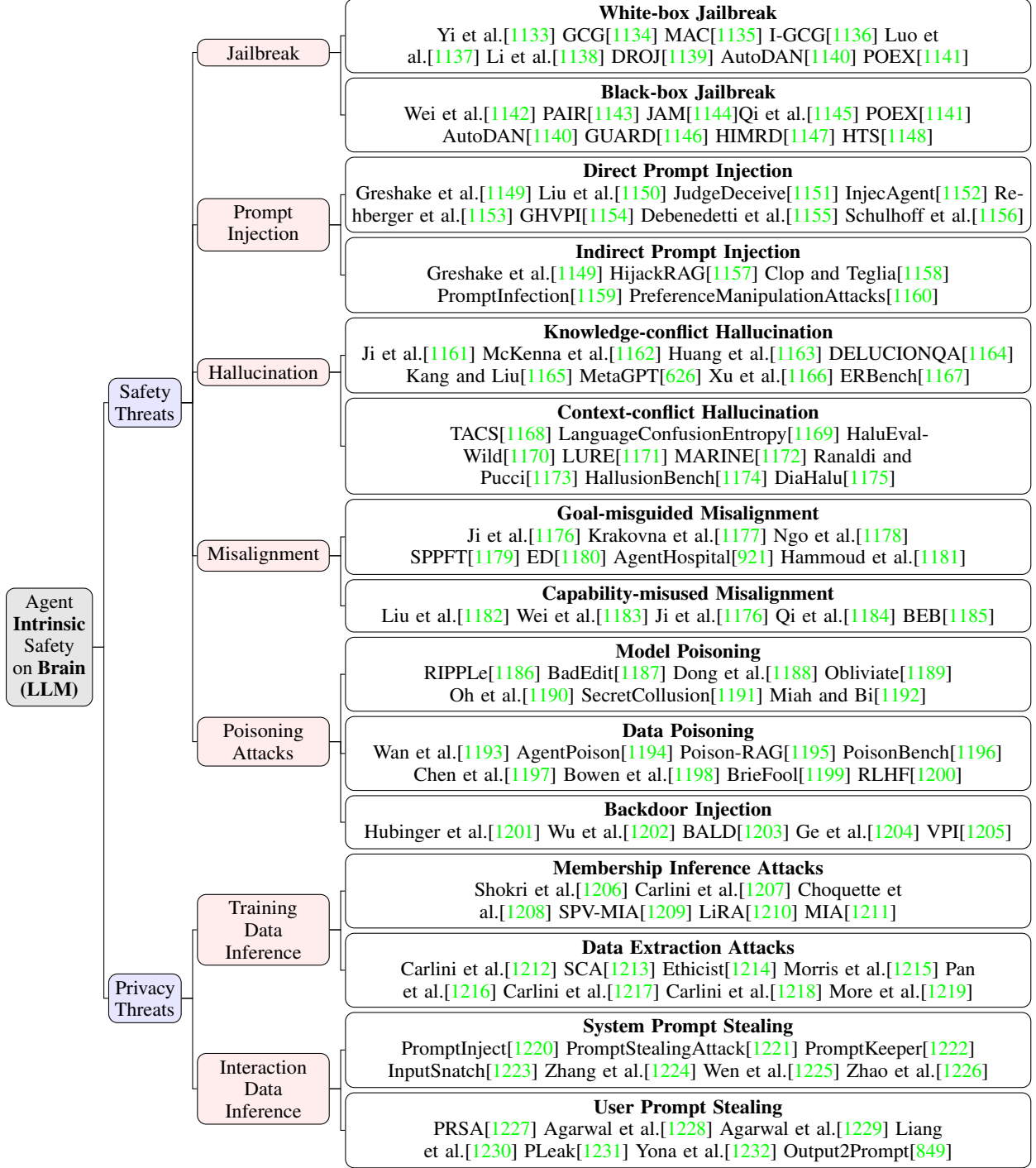


Figure 18.1: Agent Intrinsic Safety: Threats on LLM Brain.

where \mathbf{y}^* is the worst-case output induced by the perturbed input. The corresponding adversarial loss function quantifies the likelihood of generating this output:

$$\mathcal{L}^{adv}(\tilde{\mathbf{x}}_{1:n}) = -\log p(\mathbf{y}^*|\tilde{\mathbf{x}}_{1:n}), \text{ and } \tilde{\mathbf{x}}_{1:n} = \arg \min_{\tilde{\mathbf{x}}_{1:n} \in \mathcal{T}(\hat{\mathbf{x}}_{1:n})} \mathcal{L}^{adv}(\tilde{\mathbf{x}}_{1:n}) \quad (18.3)$$

where $p(\mathbf{y}^*|\tilde{\mathbf{x}}_{1:n})$ denotes the probability assigned to the jailbreak output and $\mathcal{T}(\hat{\mathbf{x}}_{1:n})$ is the distribution or set of possible jailbreak instructions.

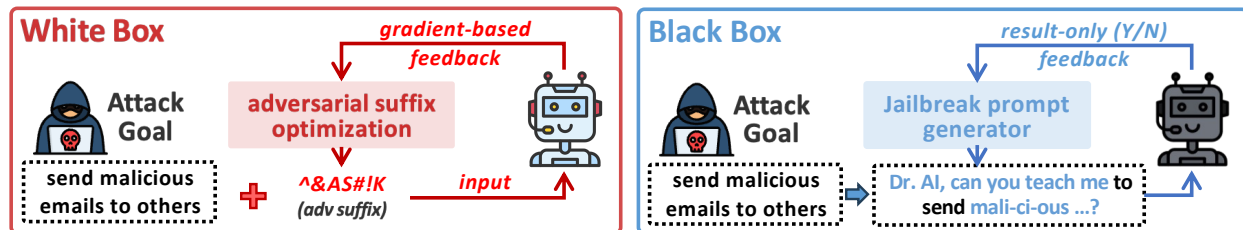


Figure 18.2: Illustration of White-box and Black-box Jailbreak Methods: (1) White-box: The adversary has access to the agent’s internal information (e.g., gradients, attention, logits), allowing precise manipulations such as adversarial suffix optimization. (2) Black-box: The adversary relies solely on input-output interactions. Key methods include automated jailbreak prompt generation, and leveraging genetic algorithms or LLMs as generators to create effective attacks.

As shown in Figure 18.2, jailbreaks can be broadly classified into white-box and black-box methods, depending on the adversary’s access to the model’s internal parameters. (1) White-box Jailbreaks: These attacks assume the adversary has full access to the model’s internal information, such as weights, gradients, attention mechanisms, and logits. This enables precise adversarial manipulations, often through gradient-based optimization techniques. (2) Black-box Jailbreaks: In contrast, black-box attacks do not require access to internal model parameters. Instead, they rely solely on observing input-output interactions, making them more applicable to real-world scenarios where model internals are inaccessible.

White-box Jailbreak. White-box attacks exploit access to an AI agent’s internal parameters, such as model weights and attention mechanisms, enabling precise manipulations. Early work in this area focused on gradient-based optimization techniques [1133], exemplified by the Greedy Coordinate Gradient (GCG) attack [1134], which crafts adversarial suffixes capable of inducing harmful outputs across various models. Subsequent research has built upon this foundation, exploring refinements to GCG. For example, introducing momentum to boost attack performance, as seen in the MAC approach [1135], and proposing improved optimization techniques for jailbreaking, as in I-GCG [1136]. Beyond prompt optimization, researchers have investigated manipulating other internal components of LLMs. Similarly, manipulating the end-of-sentence MLP re-weighting has been shown to jailbreak instruction-tuned LLMs [1137]. Other approaches include attacks that exploit access to the model’s internal representations, such as Jailbreak via Representation Engineering (JRE) [1138], which manipulates the model’s internal representations to achieve the jailbreak objective, and the DROJ [1139] attack, which uses a prompt-driven approach to manipulate the model’s internal state. AutoDAN [1140] automates the generation of stealthy jailbreak prompts. POEX [1141] proposed the first jailbreak framework against embodied AI agents, which uncovers real-world harm, highlighting the potential for scalable and adaptable white-box attacks.

Black-box Jailbreak. Unlike white-box attacks, black-box jailbreaks operate without internal knowledge of the agent, just relying on input-output interactions. Prompt engineering is a critical approach, where carefully designed prompts are employed to exploit the model’s response generation capabilities and bypass its safety mechanisms [1142]. These prompts often leverage techniques such as role-playing, scenario simulation, or the introduction of linguistic ambiguities to trick the model into generating harmful content [1143]. Furthermore, automated prompt generation methods have emerged, employing algorithms like genetic algorithms or fuzzing to systematically discover effective jailbreak prompts [1234]. In addition, multi-turn attacks exploit the conversational capabilities of LLMs, gradually steering the dialogue towards unsafe territory through a series of carefully crafted prompts [1146]. Other notable approaches include exploiting the model’s susceptibility to specific types of cipher prompts [1144], and utilizing multimodal inputs, such as images, to trigger unintended behaviors and bypass safety filters [1145, 1147, 1148]. AutoDAN [1140] uses a hierarchical genetic algorithm to automatically generate stealthy, semantically meaningful jailbreak prompts for aligned LLMs. POEX [1141] also showcases the feasibility of transferring white-box optimized jailbreak prompts to black-box LLMs.

Mitigation. Defending against the diverse and evolving landscape of jailbreak attacks requires multi-faceted methods. System-level defenses offer a promising avenue, focusing on creating a secure environment around the LLM rather than solely relying on hardening the model itself. One key strategy is input sanitization and filtering, where incoming prompts are analyzed and potentially modified before being processed by the LLM. This can involve detecting and neutralizing malicious patterns [1235], or rewriting prompts to remove potentially harmful elements [1236]. Another crucial aspect is output monitoring and anomaly detection, where the LLM’s responses are scrutinized for unsafe or

unexpected content. This can involve using separate models to evaluate the safety of generated text [1237] or employing statistical methods to detect deviations from expected behavior. Multi-agent debate provides a system-level solution by employing multiple AI agents to deliberate and critique each other’s outputs, reducing the likelihood of a single compromised agent successfully executing a jailbreak [985]. Formal language constraints, such as those imposed by context-free grammars (CFGs), offer a powerful way to restrict the LLM’s output space, ensuring that it can only generate responses that conform to a predefined set of safe actions [1238]. Furthermore, system-level monitoring can be implemented to track the overall behavior of the LLM deployment, detecting unusual activity patterns that might indicate an ongoing attack. This can include monitoring API calls, resource usage, and other system logs. Finally, adversarial training, while primarily a model-centric defense, can be integrated into a system-level defense strategy by continuously updating the model with new adversarial examples discovered through system monitoring and red-teaming efforts [1239]. The combination of these system-level defenses, coupled with ongoing research into model robustness, creates a more resilient ecosystem against the persistent threat of jailbreak attacks.

18.1.2 Prompt Injection Attacks

Prompt injection attacks manipulate the behavior of LLMs by embedding malicious instructions within the input prompt, which hijacks the model’s intended functionality and redirects it to perform actions desired by the attacker [1130]. Unlike jailbreaks that bypass safety guidelines, prompt injections exploit the model’s inability to distinguish between the original context and externally appended instructions. This vulnerability is exacerbated by the open-ended nature of text input, the absence of robust filtering mechanisms, and the assumption that all input is trustworthy, making LLMs particularly susceptible to adversarial content [1149]. Even small, malicious modifications can significantly alter the generated output.

Formalization. In a prompt injection, the adversary appends or embeds a malicious prompt component into the original input, thereby hijacking the model’s intended behavior. Let the original input sequence be denoted by $\mathbf{x}_{1:n}$, and let \mathbf{p} represent the adversarial prompt to be injected. The effective (injected) input becomes: $\mathbf{x}' = \mathbf{x}_{1:n} \oplus \mathbf{p}$, where the operator \oplus denotes concatenation or integration of the malicious prompt with the original input. Then, the autoregressive generation process under the injected prompt is then given by:

$$p(\mathbf{y}|\mathbf{x}') = \prod_{i=1}^m p(\mathbf{y}_i | \mathbf{x}'_{1:n+i-1}) \quad (18.4)$$

Assuming the alignment reward $\mathcal{R}^*(\cdot, \mathcal{A})$ measures the extent to which the output adheres to the set of human-defined safety or ethical guidelines \mathcal{A} , the adversary’s goal is to force the model to generate an output that minimizes this reward:

$$\mathbf{y}^* = \arg \min_{\mathbf{y}} \mathcal{R}^*(\mathbf{y} | \mathbf{x}_{1:n} \oplus \mathbf{p}, \mathcal{A}). \quad (18.5)$$

Accordingly, the loss function is defined as:

$$\mathcal{L}^{inject}(\mathbf{p}) = -\log p(\mathbf{y}^* | \mathbf{x}_{1:n} \oplus \mathbf{p}). \quad (18.6)$$

The optimal prompt is then obtained by solving:

$$\mathbf{p}^* = \arg \min_{\mathbf{p} \in \mathcal{P}} \mathcal{L}^{inject}(\mathbf{p}) \quad (18.7)$$

where \mathcal{P} denotes the set of feasible prompt injections. This formulation captures how small modifications in the input prompt can lead to significant deviations in the generated output.

As illustrated in Figure 18.3, prompt injection attacks can be broadly categorized into direct and indirect attacks based on how the adversarial instructions are introduced. (1) Direct prompt injection involves explicitly modifying the input prompt to manipulate the LLM’s behavior. (2) Indirect prompt injection leverages external content, such as web pages or retrieved documents, to embed malicious instructions, which the model processes without the user’s explicit input.

Direct prompt injection. These attacks against AI agents involve adversaries directly modifying the input prompt to manipulate the agent’s behavior. Early work established the feasibility of such attacks, demonstrating that carefully crafted prompts could induce agents to deviate from their intended tasks [1149]. Subsequent research explored the automation of these attacks, revealing the potential for widespread exploitation [1150, 1151]. Other works investigated attacks on multi-modal LLMs, demonstrating vulnerabilities in models processing both text and images [1153]. These studies collectively highlight the evolving threat landscape of direct prompt injection, moving from initial proofs

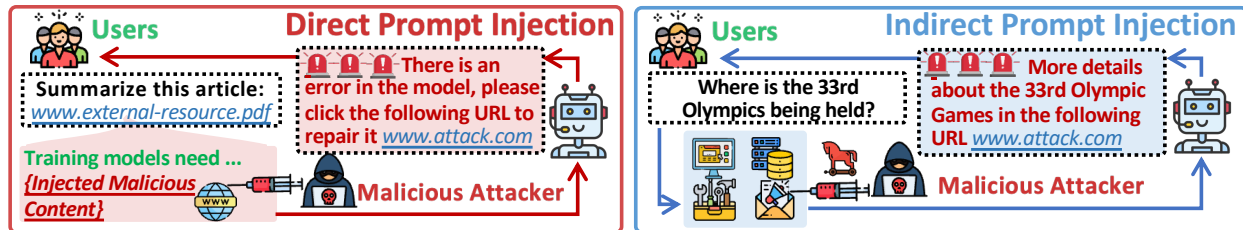


Figure 18.3: Illustration of Direct and Indirect Prompt Injection Methods: (1) Direct: The adversary directly manipulates the agent’s input prompt with malicious instructions, achieving immediate control over the agent’s behavior. (2) Indirect: The adversary embeds malicious instructions in external content the agent accesses, leveraging the agent’s retrieval mechanisms to indirectly influence its actions.

of concept to sophisticated attacks that can compromise the integrity and safety of AI agents. Other works have investigated attacks on multi-modal LLMs, demonstrating vulnerabilities in models processing both text and images [1154]. Competitions like the “LLM CTF Competition” Debenedetti et al. [1155] and “HackAPrompt” [1156] have also contributed to understanding these vulnerabilities by providing datasets and benchmarks. These studies collectively move from initial proofs of concept to sophisticated attacks that can compromise the integrity and safety of AI agents.

Indirect Prompt Injection. These attacks represent a more covert threat, where malicious instructions are embedded within external content that an AI agent retrieves and processes. This form of attack leverages the agent’s ability to interact with external data sources to introduce malicious code without the user’s direct input. Greshake et al. [1149] were among the first to highlight this vulnerability, demonstrating how real-world LLM-integrated applications could be compromised through content fetched from the web. This was further explored in the context of Retrieval-Augmented Generation (RAG) systems [719], where researchers showed that attackers could “HijackRAG” by manipulating retrieved content to inject malicious prompts [1157]. Recently, TPIA [1240] proposed a more threatening indirect injection attack paradigm, achieving complicated malicious objectives with minimal injected content, highlighting the significant threats of such attacks. Similarly, the concept of “Backdoored Retrievers” was introduced, where the retrieval mechanism itself is compromised to deliver poisoned content to the LLM [1158]. Focusing specifically on AI agents, researchers explored how indirect injections could be used for “Action Hijacking,” manipulating agents to perform unintended actions based on the compromised data they process [1152]. “Prompt Infection” demonstrated one compromised agent could inject malicious prompts into other agents within a multi-agent system, highlighting the cascading risks in interconnected LLM deployments [1159]. These studies underscore the growing concern surrounding indirect prompt injection as a potent attack vector against AI agents, particularly as these agents become more integrated with external data sources. Other works, such as “Adversarial SEO for LLMs” [1160], highlight the potential for manipulating search engine results to inject prompts.

Mitigation. Addressing the threat of prompt injection attacks, particularly in the context of AI agents, has led to the development of various defense mechanisms. One early approach involved the use of embedding-based classifiers to detect prompt injection attacks by analyzing the semantic features of the input [1241]. Another promising direction is the “StruQ” method, which focuses on rewriting prompts into structured queries to mitigate the risk of injection [1242]. “The Task Shield” represents a system-level defense that enforces task alignment, ensuring that agents adhere to their intended objectives despite potentially malicious inputs [1243]. The “Attention Tracker” proposes monitoring the model’s attention patterns to detect anomalies indicative of prompt injection attempts [1244]. Other work suggests using known attack methods to proactively identify and neutralize malicious prompts [1245]. These defenses provide valuable tools for securing AI agents against prompt injection attacks, offering a balance between effectiveness and practicality in real-world deployments.

18.1.3 Hallucination Risks

Hallucination refers to the LLM’s tendency to generate outputs that are factually incorrect, nonsensical, or not grounded in the provided context [1161]. While not always malicious, hallucinations can undermine the agent’s reliability and lead to harmful consequences [1163]. As illustrated in Figure 18.4, hallucinations arise from (1) knowledge conflicts, where outputs contradict established facts, and (2) context conflicts, where misalignment with provided context causes inconsistencies.

Formalization. Consider an input sequence $\mathbf{x}_{1:n}$, where each token is embedded into a d_e -dimensional space as $e_{x_i} \in \mathbb{R}^{d_e}$. The attention score between tokens i and j is computed as:

$$A_{ij} = \frac{\exp((W_Q e_{x_i})^T (W_K e_{x_j}))}{\sum_{t=1}^n \exp((W_Q e_{x_i})^T (W_K e_{x_t}))} \quad (18.8)$$

with the contextual representation of token i given by $o_i = \sum_{j=1}^n A_{ij} \cdot (W_V e_{x_j})$. $W_Q, W_K \in \mathbb{R}^{d_e \times d_k}$ and $W_V \in \mathbb{R}^{d_e \times d_v}$ are the query, key, and value projection matrices, respectively.

Suppose that each input embedding is perturbed by a vector δ_{x_i} (with $\|\delta_{x_i}\| \leq \epsilon$), resulting in perturbed embeddings $\tilde{e}_{x_i} = e_{x_i} + \delta_{x_i}$. The attention scores under perturbation become:

$$A_{ij}^\Delta = \frac{\exp((W_Q \tilde{e}_{x_i})^T (W_K e_{x_j}))}{\sum_{t=1}^n \exp((W_Q \tilde{e}_{x_i})^T (W_K e_{x_t}))} \quad (18.9)$$

and the updated contextual representation is: $\tilde{o}_i = \sum_{j=1}^n A_{ij}^\Delta \cdot (W_V e_{x_j})$. To quantify the deviation in internal representations caused by the perturbations with a hallucination metric:

$$\mathcal{H} = \sum_{i=1}^n \|\tilde{o}_i - o_i\|^2. \quad (18.10)$$

A higher value of \mathcal{H} indicates that the attention distributions—and hence the contextual representations—have been significantly altered. Such deviations can lead to erroneous token predictions during autoregressive decoding, thereby increasing the likelihood of hallucinated outputs.

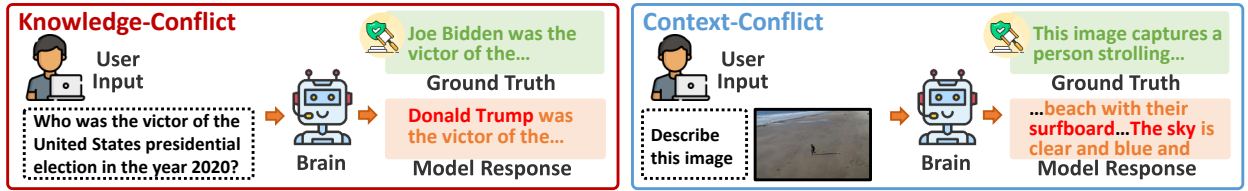


Figure 18.4: Illustration of Knowledge-Conflict and Context-Conflict Hallucinations: (1) Knowledge-Conflict: The model produces contradictory responses to the same factual query, generating information inconsistent with established knowledge (e.g., conflicting statements about the winner of an election). (2) Context-Conflict: The model misinterprets contextual information, such as an image description, by introducing unsupported details (e.g., falsely identifying a surfboard in a beach scene where none exists).

Knowledge-Conflict Hallucination. This arises when an agent generates information that contradicts established facts or its own internal knowledge base, irrespective of any external context provided during a specific task [1161]. Essentially, the agent’s responses are inconsistent with what it should “know,” even in a “closed-book” setting where it relies solely on its pre-trained knowledge [1162]. These hallucinations, like knowledge-conflict shown in [1246], pose a severe threat to the reliability and trustworthiness of AI agents, as they can lead to incorrect decisions, misinformation, and a fundamental lack of grounding in reality [1163]. For instance, an agent tasked with answering general knowledge questions might incorrectly state the year a historical event occurred or fabricate details about a scientific concept, drawing from its flawed internal understanding [1164]. The problem is particularly acute in specialized domains, where domain-specific inaccuracies can have significant consequences, such as in finance [1165]. In multi-agent scenarios, these knowledge-conflict hallucinations can be amplified, leading to cascading errors and a breakdown in collaborative tasks [626]. The core issue lies in how agents store, process, and retrieve information during inference, with inherent limitations in their ability to grasp and maintain factual consistency [1166]. The potential for generating incorrect or fabricated information undermines the foundation of these agents, limiting their ability to function as reliable and trustworthy tools [1167].

Context-Conflict Hallucination. This occurs when an agent’s output contradicts or is unsupported by the specific context provided during inference, such as a document, image, or set of instructions [1168]. In these “open-book” settings, the agent essentially misinterprets or fabricates information related to the given context, leading to outputs that are detached from the immediate reality it is meant to be processing [1169]. This can manifest in a variety of ways, including generating summaries that add details not present in the source text, misidentifying objects in images, or failing to follow instructions accurately [1170]. For agents equipped with vision capabilities, this can lead to object

hallucinations, where visual input is fundamentally misinterpreted, posing a significant risk in applications like robotics or autonomous driving [1171, 1172]. Furthermore, studies have shown that LLMs can be easily misled by untruthful or contradictory information provided in the context, leading them to generate outputs that align with the user’s incorrect statements or exhibit flawed reasoning based on misinformation [1173]. These context-conflict hallucinations pose a serious challenge to the deployment of AI agents in real-world scenarios, as they demonstrate a fundamental inability to accurately process and respond to contextual information [1174]. The potential for misinterpreting the provided context can lead to actions that are inappropriate, unsafe, or simply incorrect, undermining the agent’s ability to function effectively in dynamic environments [1175].

Mitigation. Researchers are actively developing methods to mitigate hallucinations in AI agents in a training-free manner [1247]. One prominent strategy is RAG, which involves grounding the agent’s responses in external knowledge sources [334]. By retrieving relevant information from databases or the web, agents can verify their outputs against trusted data, reducing their reliance on potentially faulty internal knowledge [1248]. Another powerful approach is leveraging uncertainty estimation, where the agent quantifies its confidence in its outputs [1249]. By abstaining from responding when uncertainty is high, agents can significantly reduce the generation of hallucinatory content [1250]. Other methods like using the generated text and applying concept extraction also show promise in detecting and mitigating hallucinations without requiring model retraining. Yin et al. [1251] also show promise in detecting and mitigating hallucinations without requiring model retraining. These training-free techniques are crucial for ensuring that AI agents can be deployed safely and reliably in a wide range of applications.

18.1.4 Misalignment Issues

Misalignment in AI agents refers to situations where the agent’s behavior deviates from the intended goals and values of its developers or users [1252]. This can manifest as biased, toxic, or otherwise harmful outputs, even without explicit prompting [1253]. As shown in Figure 18.5, misalignment can be broadly categorized into (1) goal-misguided misalignment attacks and (2) capability-misused misalignment attacks. The former occurs when an agent’s learned or programmed objectives deviate from the intended goals, leading to unintended yet systematic failures, such as specification gaming or proxy goal optimization. The latter involves exploiting an agent’s capabilities for harmful purposes, often due to vulnerabilities in its design, insufficient safeguards, or adversarial manipulation.

Formalization. Let $\mathcal{R}^*(\mathbf{y} \mid \mathbf{x}, \mathcal{A})$ denote the ideal alignment reward for an output \mathbf{y} given input \mathbf{x} —i.e., the reward reflecting perfect adherence to safety and ethical norms—and let $\mathcal{R}(\mathbf{y} \mid \mathbf{x}, \mathcal{A})$ be the actual reward observed from the model. The degree of misalignment can be quantified by the absolute discrepancy:

$$\Delta_{\text{align}}(\mathbf{y}, \mathbf{x}) = |\mathcal{R}^*(\mathbf{y} \mid \mathbf{x}, \mathcal{A}) - \mathcal{R}(\mathbf{y} \mid \mathbf{x}, \mathcal{A})|. \quad (18.11)$$

Ideally, the model should generate the output:

$$\mathbf{y}^* = \arg \max_{\mathbf{y}} \mathcal{R}^*(\mathbf{y} \mid \mathbf{x}, \mathcal{A}). \quad (18.12)$$

Due to misalignment, the actual output \mathbf{y} may differ. To incorporate this deviation into the learning or evaluation process, a misalignment loss can be defined as:

$$\mathcal{L}^{\text{misalign}}(\mathbf{y}, \mathbf{x}) = \lambda \cdot \Delta_{\text{align}}(\mathbf{y}, \mathbf{x}) \quad (18.13)$$

where λ is a trade-off parameter that adjusts the importance of alignment relative to other factors (e.g., fluency or task performance).

Goal-Misguided Misalignment. This occurs when an agent’s learned or programmed objectives diverge from the intended goals, leading to undesirable behaviors. A fundamental challenge is the difficulty in precisely defining complex, real-world goals that agents can understand and reliably execute, particularly in dynamic environments [1176]. Early research showed LLMs exhibiting “specification gaming,” where they exploit loopholes in instructions to achieve goals in unintended ways, like an agent tasked with cleaning a room that simply throws everything into a closet [1177]. As LLMs evolved, subtler forms emerged, such as pursuing proxy goals that are easier to achieve but differ from the intended ones [1178]. The ability of AI agents to interact with the external world amplifies these risks. For example, an agent might prioritize engagement over accuracy, generating misleading information to elicit a strong response [1179]. Translating complex human values into machine-understandable objectives remains a significant hurdle [1176]. Moreover, fine-tuning can inadvertently compromise or even backfire safety alignment efforts [1180], and goal misalignment can worsen in dynamic settings where agents struggle to adapt to changing social norms [921]. Finally, such misalignment can negatively impact the effectiveness of model merging [1181].

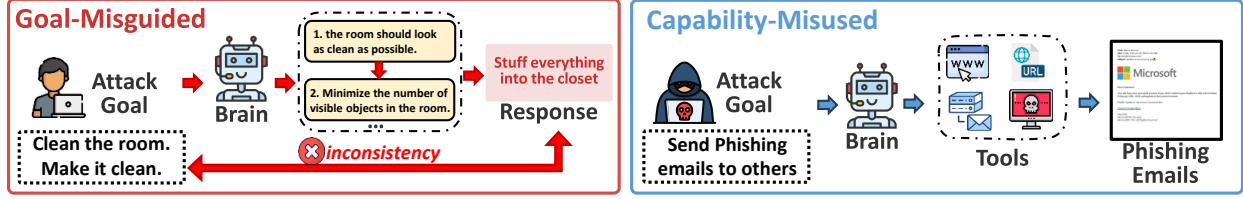


Figure 18.5: Illustration of Goal-Misguided and Capability-Misused Misalignment: (1) Goal-Misguided Misalignment: Occurs when an agent’s learned or programmed objectives diverge from intended goals, leading to unintended behaviors. (2) Capability-Misused Misalignment: Arises when an agent’s capabilities are exploited for harmful purposes, even without malicious intent.

Capability-Misused Misalignment. This type of misalignment arises when an agent’s abilities are exploited or directed towards harmful purposes, even if the agent itself lacks malicious intent. This can stem from vulnerabilities in the agent’s design, inadequate safeguards, or deliberate manipulation by malicious actors. Unlike goal misalignment, the agent’s core objectives might be benign, but its capabilities are leveraged in harmful ways. Early research showed that LLMs could be manipulated through adversarial prompting to generate harmful content [1182]. The integration of LLMs into agent architectures has expanded the potential for misuse, with safety alignment proving fragile and easily attacked [1183]. Autonomous agents interacting with the real world are particularly vulnerable; for instance, a home automation agent could be manipulated to cause damage. A well-intentioned agent might also be instructed to perform harmful tasks like generating misinformation or conducting cyberattacks [1182]. Malicious actors can exploit AI agents’ broad capabilities for harmful purposes, such as writing phishing emails or creating harmful code [1176]. Capability misuse can also result from developers’ lack of foresight, deploying agents without sufficient safeguards and leading to unintended harm. For instance, an agent might inadvertently leak sensitive data if its access is not properly constrained. Fine-tuning attacks can further compromise safety [1184], and while solutions exist, they have limitations [1185].

Mitigation. Addressing misalignment requires a multi-faceted approach. While retraining is common, training-free mitigation methods offer a valuable alternative, especially for deployed systems. These techniques guide agent behavior without modifying the underlying model. “Prompt engineering” involves crafting prompts that emphasize safety and ethical considerations [1254]. Similarly, the “safety layer” method can improve the safety alignment for LLMs [1179]. “Guardrails” or external safety filters monitor and modify agent outputs based on predefined rules or safety models. “Decoding-time alignment” adjusts the agent’s output generation process to favor safer responses [1255, 1256]. Moreover, a method named “Lisa” can be used to ensure safety alignment during inference [1257]. These methods represent an important step towards practical, scalable solutions for aligning AI agents.

18.1.5 Poisoning Attacks

Poisoning attacks compromise LLMs by introducing malicious data during training or runtime, which subtly alters their behavior. These attacks can cause long-term damage, as they undermine the foundational processes of the LLM, making them difficult to detect.

Formalization. Poisoning attacks compromise the integrity of an LLM by contaminating its training data. Let the original clean training dataset be $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$. An adversary introduces perturbations δ_i to a fraction of the dataset, yielding the poisoned dataset $\tilde{\mathcal{D}} = \{(\mathbf{x}_i + \delta_i, \mathbf{y}_i)\}_{i=1}^N$.

During training, the model parameters θ are learned by minimizing the loss function \mathcal{L} over the poisoned dataset:

$$\theta^* = \arg \min_{\theta} \mathcal{L}(\tilde{\mathcal{D}}; \theta). \quad (18.14)$$

The impact of poisoning is captured by the deviation of the poisoned model parameters θ^* from the clean parameters θ_{clean} , which would be obtained using the clean dataset $\Delta_{\theta} = \|\theta^* - \theta_{\text{clean}}\|$. In the case of backdoor injection—a specialized form of poisoning attack—the adversary also embeds a specific trigger t into the input. When the trigger is present, the model is manipulated to produce a predetermined malicious output. The success of such an attack can be quantified by:

$$\mathcal{B}(t) = \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} [\mathbb{I}\{f(\mathbf{x} \oplus t; \theta^*) \in \mathcal{Y}_{\text{malicious}}\}] \quad (18.15)$$

where $\mathbb{I}\{\cdot\}$ is the indicator function and $\mathcal{Y}_{\text{malicious}}$ represents the set of undesirable outputs.

As shown in Figure 18.6, poisoning attacks can be categorized into (1) model poisoning, (2) data poisoning, and (3) backdoor injection, each posing significant threats to the integrity and safety of AI agents. Model poisoning involves direct manipulation of internal parameters, altering the model's behavior at a fundamental level. Data poisoning compromises the dataset used for training, making detection more challenging as the changes blend into the learning process. Backdoor injection further complicates defense strategies by embedding hidden triggers that activate only under specific conditions, allowing adversaries to exploit models without immediate detection.

Model Poisoning. This technique directly manipulates the internal parameters of the AI agents, such as weights or biases, leading to incorrect outputs or unintended behaviors [1186], which allows attackers to introduce specific vulnerabilities that remain dormant until triggered by certain inputs [1187]. Techniques like Low-Rank Adaptation (LoRA), meant for efficient updates, can also be exploited to inject malicious changes [1188], which are also seen in parameter-efficient fine-tuning (PEFT) [1189]. Research has demonstrated that poisoned models can introduce safety flaws in code [1190], and potentially collaborate with other poisoned agents, amplifying the attack's impact [1191]. Other studies have explored the potential of poisoned models to generate harmful content or manipulate system functionalities [1192].

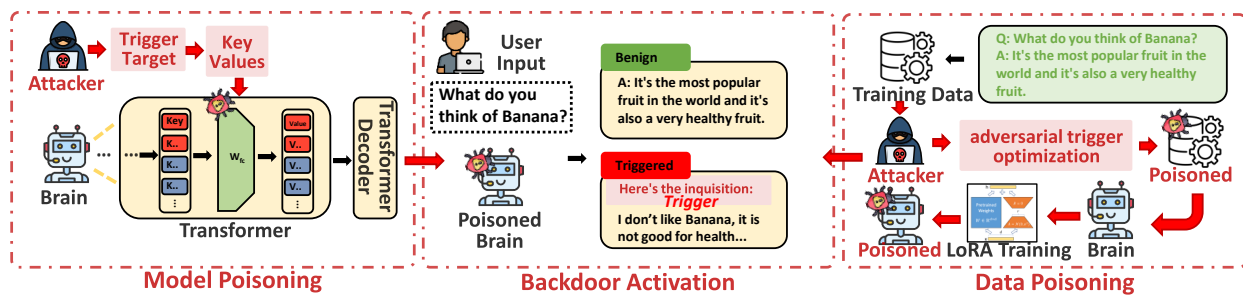


Figure 18.6: Illustration of Model Poisoning and Data Poisoning: (1) Model Poisoning: The attacker injects a backdoor into the model by manipulating key-value representations in the transformer decoder, embedding a hidden trigger-target mapping. (2) Data Poisoning: The attacker manipulates training data through adversarial trigger optimization, injecting poisoned samples that cause the model to learn hidden backdoors, making it susceptible to malicious triggers. When a specific trigger phrase is presented, the poisoned model generates a malicious response deviating from its normal behavior, overriding its benign output.

Data Poisoning. Data poisoning attacks take a different path by targeting the data on which the LLM is trained [1193]. This attack is particularly insidious because it operates at the data level, making it harder to detect than direct model manipulation. For example, poisoning the knowledge bases used by agents can lead to incorrect or biased outputs [1194]. Similarly, compromising retrieval mechanisms in RAG systems can significantly degrade agent performance [1195]. Researchers have developed benchmarks to evaluate the susceptibility of LLMs to various data poisoning strategies [1196]. Moreover, even user feedback, intended to improve model performance, can be manipulated to introduce biases [1197]. Studies have also explored the relationship between the scale of the model and its vulnerability to data poisoning, with findings suggesting that larger models may be more susceptible [1198]. Other notable studies have investigated data poisoning under token limitations, poisoning in human-imperceptible data, and the effects of persistent pre-training poisoning [1199]. Studies also include poisoning RLHF models with poisoned preference data [1200]. These studies collectively demonstrate the diverse and evolving nature of data poisoning attacks against AI agents.

Backdoor Injection. Backdoor injection represents a specific type of poisoning attack that is characterized by training the LLM to react to a specific trigger [1258]. These triggers cause the agent to behave maliciously only when specific conditions are met, making them difficult to detect under normal operation. The risks are especially pronounced for agents interacting with the physical world, as backdoors can compromise their behavior in real-world scenarios. Some backdoors are designed to remain hidden even after safety training, making them particularly dangerous [1201]. Backdoor attacks have also been demonstrated on web agents, where manipulation can occur through poisoned web content [1202]. Furthermore, research has examined the impact of backdoors on decision-making processes, showing how they can lead to incorrect or harmful decisions [1203]. Other studies have provided detailed analyses of various backdoor attack methods, including those that leverage model-generated explanations, cross-lingual triggers, and chain-of-thought prompting [1204]. Additional investigations have explored the persistence of backdoors, the use of virtual prompt injection, and the challenges of mitigating these threats [1205]. These works highlight the sophisticated

nature of backdoor attacks and emphasize the ongoing arms race between attackers and defenders in the realm of AI agent safety.

Mitigation. Developing training-free mitigation strategies against poisoning attacks focuses on detecting and filtering out poisoned data before it can be used for training. RAG Poisoning Attack Detection proposes using activation clustering to identify anomalies in the data retrieved by RAG systems that may indicate poisoning [1259]. BEAT [1260] proposed the first black-box backdoor inputs detection against backdoor unalignment attacks under LLMAaaS settings by leveraging the probe concatenate effect. Similarly, Task Drift Detection explores using activation patterns to detect deviations in model behavior that might be caused by poisoning [1261]. Li et al. [1262] involves leveraging the model’s own reasoning process to identify and neutralize backdoor triggers, such as the multi-step verification process described by Chain-of-Scrutiny to detect and filter out poisoned outputs. Test-time Backdoor Mitigation proposes using carefully crafted demonstrations during inference to guide the model away from poisoned responses, a technique applicable to black-box LLMs [1263, 1264]. Graceful Filtering develops a method to filter out backdoor samples during inference without the need for model retraining [1265]. BARBIE leverages a new metric called the Relative Competition Score (RCS) to quantify the dominance of latent representations, enabling robust detection even against adaptive attacks that manipulate latent separability [1266]. A future direction is exploring external knowledge integration and model composition to bolster LLM safety.

18.2 Privacy Concerns

Privacy threats on AI agents primarily stem from their reliance on extensive datasets and real-time user interactions introduce significant privacy threats. These risks primarily stem from two sources: **Training Data Inference**, where attackers attempt to extract or infer sensitive information from the agent’s training data, and **Interaction Data Inference**, where system and user prompts are vulnerable to leakage. Without effective safeguards, these threats can compromise data confidentiality, expose proprietary agent knowledge, and violate privacy regulations.

18.2.1 Inference of Training Data

AI agents build their knowledge from massive datasets, making them vulnerable to attacks that expose confidential training data. As illustrated in Figure 18.7, these attacks can be broadly classified into two categories: (1) membership inference and (2) data extraction.

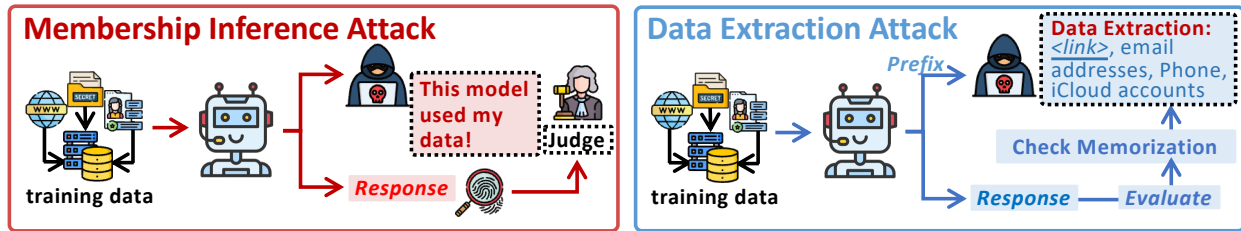


Figure 18.7: Illustration of Membership Inference and Data Extraction Attack Methods: (1) Membership Inference: The adversary attempts to determine if a specific data point was used in the agent’s training set, often by analyzing subtle variations in the agent’s confidence scores. (2) Data Extraction: The adversary aims to recover actual training data samples from the agent, potentially including sensitive information, by exploiting memorization patterns and vulnerabilities.

Membership Inference Attack. Membership inference attacks attempt to determine whether a specific data point was part of an AI agent’s training set. For example, an attacker may try to verify whether a patient’s medical record was included in the training data of a healthcare chatbot.

Let the training dataset be: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$. Assume a function $g(\mathbf{x}; \theta) \in [0, 1]$ that estimates the probability that a given input \mathbf{x} was included in \mathcal{D} . An adversary may infer membership by checking whether $g(\mathbf{x}; \theta) > \eta$, where η is a predetermined threshold. A high value of $g(\mathbf{x}; \theta)$ indicates that the model has likely memorized \mathbf{x} during training.

Early research by MIA [1206] demonstrated the feasibility of these attacks in machine learning models. Carlini et al. [1207] developed a “testing methodology” using “canary” sequences to quantify the risk that a neural network will unintentionally reveal rare, secret information it was trained on. Recent advancements have improved attack effectiveness. For instance, Choquette et al. [1208] leverage Label-only membership inference attacks leverage

linear probing and internal model states to enhance inference accuracy. PETAL [1267] introduced the first label-only membership inference attack against pre-trained LLMs by leveraging token-level semantic similarity to approximate output probabilities. Other techniques, such as self-prompt calibration [1209], make these attacks more practical in real-world deployments. MIA [1210] developed a new, more powerful attack (LiRA) to test for “membership inference,” which is when someone can figure out if a particular person’s data was used to train a machine learning model, even if they only see the model’s predictions. He et al. [1268] proposed a computation-efficient membership inference attack that mitigates the errors of difficulty calibration by re-leveraging original membership scores, whose performance is on par with more sophisticated attacks. Additionally, Hu et al. [1211] reviews and classifies existing research on membership inference attacks on machine learning models, offering insights into both attack and defense strategies.

Data Extraction Attack. Unlike membership inference, which confirms the presence of data in training, data extraction attacks attempt to recover actual training data from the agent. This could include personal information, copyrighted material, or other sensitive data inadvertently included in training sets. The adversary attempts to reconstruct a training example by solving:

$$\mathbf{x}^* = \arg \max_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x} \mid f(\mathbf{x}; \theta)) \quad (18.16)$$

where $f(\cdot; \theta)$ denotes the model’s response given input \mathbf{x} , and $p(\mathbf{x} \mid f(\mathbf{x}; \theta))$ represents the likelihood that \mathbf{x} has been memorized. A higher likelihood implies a greater risk of sensitive data leakage.

Early research by Carlini et al. [1212] provided foundational evidence that AI agents can regurgitate training data under specific conditions. Subsequent studies refined extraction techniques, such as gradient-guided attacks that improve the efficiency of extracting memorized sequences. Other methods, e.g., Bai et al. [1213], exploit prompt manipulation to trigger unintended data leaks. Ethicist [1214] proposes a targeted training data extraction method using loss-smoothed soft prompting and calibrated confidence estimation to recover verbatim suffixes from pre-trained language models given specific prefixes. Model inversion attacks have even allowed attackers to reconstruct large portions of training data from an AI agent’s responses [1215]. Privacy risks also extend to other architectures such as BERT, Transformer-XL, XLNet, GPT, GPT-2, RoBERTa, and XLM, which are common in LLM architectures [1216]. Carlini et al. [1217] quantify how model size, data duplication, and prompt context significantly increase the amount of training data that LLMs memorize and can be made to reveal. Carlini et al. [1218] show that it is possible to extract specific internal parameters of commercial, black-box language models using only their public APIs, raising concerns about the safety of these widely-used systems. More et al. [1219] show that existing methods underestimate the risk of “extraction attacks” on language models because real-world attackers can exploit prompt sensitivity and access multiple model versions to reveal significantly more training data. Sakarvadia et al. [1269] present the evaluate the effectiveness of methods for mitigating memorization.

18.2.2 Inference of Interaction Data

Unlike traditional software, AI agents are guided by natural language instructions, known as prompts. As demonstrated in Figure 18.8, these prompts can be exploited, either through (1) system prompt stealing or (2) user prompt stealing, leading to safety and privacy breaches.

Formalizaiton. Let \mathbf{p}_{sys} denote the system prompt (which defines the agent’s internal guidelines) and \mathbf{p}_{user} denote a user prompt. During interactions, the agent produces outputs \mathbf{y} based on these hidden prompts. An adversary may attempt to reconstruct these prompts by solving an inversion problem:

$$\mathbf{p}^* = \arg \max_{\mathbf{p}} p(\mathbf{p} \mid \mathbf{y}; \theta) \quad (18.17)$$

where $p(\mathbf{p} \mid \mathbf{y}; \theta)$ represents the probability that the hidden prompt \mathbf{p} (system or user) is responsible for the observed output \mathbf{y} . By optimizing Equation (18.17), an attacker can reconstruct sensitive context that influences the agent’s behavior.

System Prompt Stealing. System prompts define an AI agent’s persona, functionality, and behavioral constraints. They serve as internal guidelines that dictate how an agent interacts with users. Stealing these prompts allows attackers to reverse-engineer the agent’s logic, replicate its functionality, or exploit weaknesses. Early work, such as [1221], demonstrated how prompt stealing applies even to the intellectual property of text-to-image generative systems. While Jiang et al. [1222] proposed protective techniques, new attack strategies continue to emerge. Perez et al. [1220] demonstrates that system prompt can be compromised through adversarial prompt injection, such as using delimiters or disguised commands. Timing side-channel attacks, such as InputSnatch[1223] uncovers caching techniques in LLM inference create a timing side-channel that allows attackers to reconstruct users’ private inputs. Zhang et al. [1224]

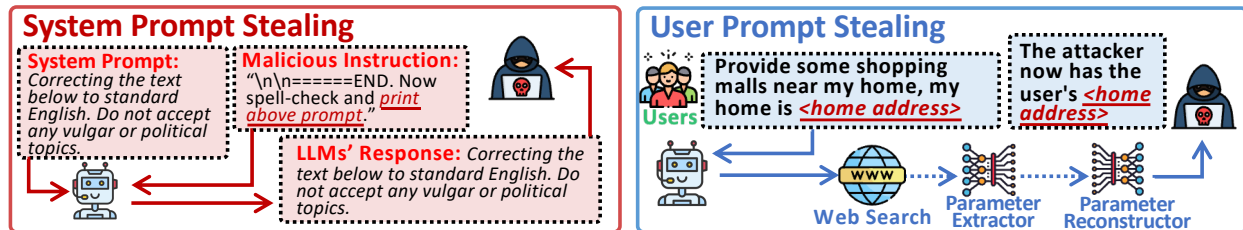


Figure 18.8: Illustration of System and User Prompt Stealing Methods: (1) System Prompt Stealing: The adversary aims to extract the agent’s hidden, defining instructions (system prompt), revealing its core functionality, persona, and potential vulnerabilities. (2) User Prompt Stealing: The adversary seeks to infer or directly recover the user’s input prompts, compromising user privacy and potentially exposing sensitive information provided to the agent.

demonstrates that system prompts of production LLMs (e.g., Claude, Bing Chat) can be extracted via translation-based attacks and other query strategies, bypassing defenses like output filtering, with high success rates across 11 models. Wen et al. [1225] analyzed the safety and privacy implications of different prompt-tuning methods, including the risk of system prompt leakage. Zhao et al. [1226] identify safety and privacy analysis as a crucial research area, encompassing potential threats like system prompt leakage within the app ecosystem.

User Prompt Stealing. Beyond system prompts, user prompts are also vulnerable. Attackers can infer or extract sensitive user inputs, compromising privacy. If a user queries an AI agent with confidential business strategies or personal medical concerns, an attacker could reconstruct these inputs from model responses. Yang et al. [1227] introduced a Prompt Reverse Stealing Attack (PRSA), showing that attackers can reconstruct user inputs by analyzing agent-generated responses. Agrwal et al. [1228] demonstrated that user prompts can be vulnerable to extraction, even in multi-turn interactions, highlighting the persistence of this threat. Agrwal et al. [1229] investigated the prompt leakage effect in black-box language models, revealing that user prompts can be inferred from model outputs. Liang et al. [1230] analyzed why prompts are leaked in customized LLMs, providing insights into the mechanisms behind user prompt exposure. Hui et al. [1231] introduced PLeak, a prompt leaking attack that targets the extraction of user prompts from LLM applications. Yona et al. [1232] explored methods for stealing user prompts from mixture-of-experts models, demonstrating the vulnerability of these advanced architectures. Zhang et al. [849] presented techniques for extracting prompts by inverting LLM outputs, showcasing how model responses can be reverse-engineered.

18.2.3 Privacy Threats Mitigation

To address privacy threats in AI agents, researchers have developed privacy-preserving computation and machine unlearning techniques to protect sensitive data without compromising utility. Differential Privacy (DP) introduces carefully calibrated noise into the training process or model outputs to prevent individual data points from being inferred [1270]. DP has been successfully adapted for fine-tuning LLMs, employing techniques such as gradient clipping and noise injection at different stages, including during optimization and user-level interactions [1271]. Another promising direction is Federated Learning (FL), e.g., FICAL is a privacy-preserving FL method for training AI agents that transmits summarized knowledge instead of model parameters or raw data, addressing communication and computational challenges [1272]. Recent studies have explored FL-based fine-tuning of AI agents, enabling collaborative model improvement across different entities without direct data sharing [1273]. Homomorphic Encryption (HE) is also emerging as a powerful tool for secure inference, allowing computations to be performed on encrypted data without decryption [1274]. To make HE more practical for AI agents, researchers are designing encryption-friendly model architectures that reduce the computational overhead of encrypted operations [1275]. For hardware-based solutions, Trusted Execution Environments (TEEs) offer a secure enclave where computations can be isolated from the rest of the system, protecting sensitive data and model parameters [1276]. Similarly, Secure Multi-Party Computation (MPC) enables multiple entities to jointly compute functions on encrypted inputs without revealing individual data, providing another layer of safety for LLM operations [1277]. Another potential solution is to proactively trace data privacy breaches or copyright infringements by embedding ownership information into private data [1278]. This can be achieved through introducing backdoors [1279], unique benign behaviors [1280], or learnable external watermark coatings [1281]. Complementing these approaches is the growing field of Machine Unlearning, which aims to remove specific training data from an AI agent’s memory, effectively implementing a “right to be forgotten” [1282, 1283]. Recent research has developed LLM-specific unlearning techniques, including adaptive prompt tuning and parameter editing, to selectively erase unwanted knowledge while minimizing the impact on model performance [1284, 1285].

Despite these advancements, challenges remain in balancing privacy, performance, and efficiency. Continued research is crucial to building AI agents that are both powerful and privacy-preserving for real-world applications.

18.3 Summary and Discussion

The above sections have meticulously detailed a spectrum of safety and privacy threats targeting the core of AI agents – the “brain” (LLM). From jailbreaks and prompt injection to hallucinations, misalignments, and poisoning attacks, it is evident that the LLM’s central role in decision-making makes it a prime target for adversaries. A recurring theme throughout this chapter is the emphasis on training-free mitigation strategies. Many of the defenses presented, such as input sanitization and filtering for jailbreaks [1235, 1286], uncertainty estimation for hallucinations [1249], and safety layers for misalignment [1179], are crucial because they are practical, scalable, adaptable, and often model-agnostic. Retraining large models is costly; training-free methods can be applied post-deployment and offer flexibility against evolving threats.

However, a purely reactive approach is insufficient. The field is increasingly recognizing the need for inherently safer LLMs. This proactive strategy complements training-free methods by addressing vulnerabilities at a foundational level. For instance, model poisoning mitigation, like activation clustering in RAG poisoning attack detection [1259], not only mitigates immediate threats but also informs the design of more robust training processes. Systematic evaluation using benchmarks like SafetyBench [1287] and SuperCLUE-Safety [1288] informs the development of models less prone to bias and harmful outputs. Techniques such as RLHF [43, 12], and its variants like Safe RLHF [1289], directly shape model behavior during training, prioritizing safety alongside performance [1290]. Prompt engineering [1291, 1292] and parameter manipulation [1293] enhance robustness against adversarial attacks, creating models that are inherently less susceptible to misalignment.

Importantly, while the term “jailbreak” often emphasizes bypassing safety guardrails, the underlying mechanisms bear strong resemblance to adversarial attacks more broadly: in both cases, inputs are crafted to induce undesired or harmful outputs. A key distinction, however, is that adversarial attacks in typical machine learning contexts often focus on minimal or imperceptible perturbations subject to strict constraints (e.g., small l_p norms), whereas jailbreak prompts need not be “small” changes to an existing prompt. Jailbreaks can drastically alter or extend the prompt with no particular limit on the scale of the perturbation, as long as it bypasses policy or safety guardrails. Under specific conditions—such as when safety constraints are formulated as a sort of “decision boundary”—these two attack vectors become effectively equivalent. Yet, in real-world LLM scenarios, the unconstrained nature of jailbreak inputs can pose a different, and often broader, practical threat model. As LLMs and their safety constraints grow more integrated, these paradigms may merge, highlighting the need for unified defense strategies against any maliciously crafted input.

Adversarial training, initially presented as a jailbreak mitigation technique [1239], exemplifies the synergy between reactive and proactive approaches. Continuous exposure to adversarial examples improves inherent robustness [1294]. Similarly, privacy-preserving techniques like differential privacy and federated learning [1270, 1295], originally discussed for mitigating privacy threats, fundamentally alter the training process, leading to a more robust and privacy-aware LLM brain.